Universidad La Salle

Facultad de Ingeniería

Ingeniería de Software

EMPLEOYA

Plataforma Web de Bolsa de Trabajo

Curso: Ingeniería Web

Autores:

Piero De La Cruz
Jerson Chura

Año 2025

Índice

1. Introducción

1.1. Descripción del Proyecto

EMPLEOYA es una plataforma web de bolsa de trabajo desarrollada con Django 5.2.7, diseñada para conectar empresas empleadoras con profesionales en búsqueda de oportunidades laborales. El sistema implementa un modelo bidireccional donde ambos actores (empleadores y postulantes) tienen funcionalidades específicas adaptadas a sus necesidades.

1.2. Problemática

En el mercado laboral actual, existe una desconexión entre:

- **Empresas:** Dificultad para encontrar candidatos calificados de manera eficiente
- **Postulantes:** Falta de plataformas centralizadas y profesionales para buscar empleo
- **Proceso:** Falta de herramientas digitales que faciliten el proceso de postulación y selección

1.3. Solución Propuesta

EMPLEOYA proporciona:

1. Sistema de autenticación con dos roles diferenciados
2. Perfiles profesionales completos con indicador de progreso
3. Publicación y gestión de ofertas laborales
4. Sistema de postulaciones con seguimiento de estados
5. Panel administrativo para supervisión del sistema
6. API REST para futuras integraciones

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar una plataforma web funcional que facilite la conexión entre empresas y postulantes mediante herramientas digitales eficientes.

1.4.2. Objetivos Específicos

- Implementar sistema de autenticación seguro con roles diferenciados
- Crear perfiles profesionales con sistema de completitud al 100 %
- Desarrollar módulo CRUD completo para ofertas de trabajo
- Implementar sistema de postulaciones con gestión de estados
- Crear panel administrativo con visualización mejorada
- Desarrollar API REST documentada para integraciones futuras

2. Tecnologías Utilizadas

2.1. Stack Tecnológico

Tecnología	Versión	Propósito
Django	5.2.7	Framework web backend principal
Django REST Framework	3.15.2	Desarrollo de API REST
SQLite	3.x	Base de datos en desarrollo
Python	3.10+	Lenguaje de programación
HTML5 + CSS3	-	Frontend y diseño responsive
Pillow	11.3.0	Procesamiento de imágenes
django-cors-headers	4.6.0	Gestión de CORS para API

Cuadro 1: Stack tecnológico del proyecto

2.2. Arquitectura MVT (Model-View-Template)

Django implementa el patrón arquitectónico MVT:

- **Model (Modelo):** Define la estructura de datos y lógica de negocio
- **View (Vista):** Maneja la lógica de aplicación y procesamiento de peticiones
- **Template (Plantilla):** Renderiza la interfaz de usuario

Ventajas de Django

- ORM poderoso para abstracción de base de datos
- Sistema de autenticación integrado
- Panel de administración automático
- Seguridad incorporada (CSRF, SQL Injection, XSS)
- Escalabilidad y mantenibilidad

3. Arquitectura del Sistema

3.1. Estructura del Proyecto

```
empleoya/
          MyWebApps/                # Aplicacion principal
                models.py             # Modelos de BD
                views.py              # Logica de vistas
                urls.py               # Rutas de la app
                admin.py              # Panel admin
                forms.py              # Formularios
                utils.py              # Funciones auxiliares
                api_views.py          # Endpoints API
                serializers.py        # Serializadores API
                migrations/           # Migraciones BD
                templates/            # Plantillas HTML
                    MyWebApps/
                        base.html
                        home.html
                        login.html
                        register.html
                        dashboard_postulante.html
                        dashboard_empresa.html
                        perfil_postulante.html
                        ... (m s templates)

          empleoya_django/          # Configuracion del proyecto
                settings.py           # Configuracion principal
                urls.py               # URLs principales
                wsgi.py               # WSGI config
                asgi.py               # ASGI config

          media/                    # Archivos subidos
                cvs/
                logos/
                fotos_perfil/

          manage.py                 # Script de Django
          db.sqlite3                # Base de datos
          crear_categorias.py       # Script inicial
          requirements.txt          # Dependencias
```

Listing 1: Estructura de directorios

3.2. Flujo de Datos

1. **Petición HTTP:** El cliente envía una petición al servidor
2. **URL Routing:** Django mapea la URL a una vista específica
3. **Vista:** Procesa la petición, interactúa con modelos
4. **Modelo:** Realiza operaciones en la base de datos

5. **Template:** Renderiza la respuesta HTML
6. **Respuesta HTTP:** Se envía al cliente

4. Base de Datos

4.1. Diagrama Entidad-Relación (Conceptual)

El sistema cuenta con 8 modelos principales:

1. **Usuario** - Almacena todos los usuarios del sistema
2. **Empresa** - Información de empresas empleadoras
3. **PerfilPostulante** - Datos profesionales de postulantes
4. **Categoria** - Categorías de ofertas laborales
5. **OfertaTrabajo** - Ofertas publicadas por empresas
6. **Postulacion** - Registro de postulaciones
7. **Favorito** - Ofertas marcadas como favoritas
8. **Notificacion** - Sistema de notificaciones

4.2. Descripción Detallada de Modelos

4.2.1. Modelo: Usuario

```
class Usuario(AbstractBaseUser):
    email = models.EmailField(unique=True)
    nombre = models.CharField(max_length=100)
    apellido = models.CharField(max_length=100)
    telefono = models.CharField(max_length=20)
    tipo_usuario = models.CharField(
        max_length=20,
        choices=[
            ('postulante', 'Postulante'),
            ('empleador', 'Empleador')
        ]
    )
    foto_perfil = models.ImageField(
        upload_to='fotos_perfil/',
        blank=True,
        null=True
    )
    estado = models.CharField(max_length=20)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

```

```
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['nombre']
```

Listing 2: Modelo Usuario (fragmento)

Campos principales:

- `email`: Identificador único del usuario
- `tipo_usuario`: Diferencia entre postulante y empleador
- `foto_perfil`: Imagen de perfil del usuario
- `is_staff`: Permite acceso al panel admin

4.2.2. Modelo: PerfilPostulante

```
class PerfilPostulante(models.Model):
    usuario = models.OneToOneField(
        Usuario,
        on_delete=models.CASCADE,
        related_name='perfil_postulante'
    )
    titulo_profesional = models.CharField(max_length=200)
    resumen_profesional = models.TextField()
    nivel_experiencia = models.CharField(max_length=50)
    habilidades = models.TextField()
    cv = models.FileField(upload_to='cvs/')
    linkedin_url = models.URLField(blank=True)
    github_url = models.URLField(blank=True)
    portafolio_url = models.URLField(blank=True)
    porcentaje_completado = models.IntegerField(default=0)
    completado = models.BooleanField(default=False)

    def calcular_porcentaje_completado(self):
        campos_requeridos = [
            self.titulo_profesional,
            self.resumen_profesional,
            self.nivel_experiencia,
            self.habilidades,
            self.cv,
            self.usuario.foto_perfil,
            (self.linkedin_url or self.github_url
             or self.portafolio_url),
        ]
        completados = sum(1 for campo in campos_requeridos
                          if campo)
        porcentaje = int((completados / len(campos_requeridos))
                         * 100)
        self.porcentaje_completado = porcentaje
        self.completado = porcentaje >= 70
        self.save()
        return porcentaje
```

Listing 3: Modelo PerfilPostulante (fragmento)

Características destacadas:

- Relación uno-a-uno con Usuario
- Cálculo automático del porcentaje de completitud
- Validación de 7 campos obligatorios para alcanzar 100 %
- Almacenamiento de CV y enlaces profesionales

4.2.3. Modelo: OfertaTrabajo

```
class OfertaTrabajo(models.Model):
    empresa = models.ForeignKey(Empresa,
                                on_delete=models.CASCADE)
    categoria = models.ForeignKey(Categoria,
                                  on_delete=models.SET_NULL,
                                  null=True)
    titulo = models.CharField(max_length=200)
    descripcion = models.TextField()
    requisitos = models.TextField()
    salario_min = models.DecimalField(max_digits=10,
                                      decimal_places=2)
    salario_max = models.DecimalField(max_digits=10,
                                      decimal_places=2)
    modalidad = models.CharField(max_length=50)
    tipo_contrato = models.CharField(max_length=50)
    nivel_experiencia = models.CharField(max_length=50)
    vacantes_disponibles = models.IntegerField(default=1)
    estado = models.CharField(max_length=50,
                              default='activa')
    aprobada_admin = models.BooleanField(default=True)
    vistas = models.IntegerField(default=0)
    total_postulaciones = models.IntegerField(default=0)
    fecha_publicacion = models.DateTimeField(auto_now_add=True)
    fecha_expiracion = models.DateTimeField()
```

Listing 4: Modelo OfertaTrabajo (fragmento)

4.2.4. Modelo: Postulacion

```
class Postulacion(models.Model):
    oferta = models.ForeignKey(OfertaTrabajo,
                               on_delete=models.CASCADE)
    postulante = models.ForeignKey(PerfilPostulante,
                                   on_delete=models.CASCADE)
    estado = models.CharField(
        max_length=50,
        default='pendiente',
        choices=[
            ('pendiente', 'Pendiente'),
            ('en_revision', 'En Revision'),
            ('preseleccionado', 'Preseleccionado'),
            ('entrevista', 'Entrevista'),
```

```
            ('rechazado', 'Rechazado'),
            ('aceptado', 'Aceptado'),
        ]
    )
    carta_presentacion = models.TextField(blank=True)
    fecha_postulacion = models.DateTimeField(auto_now_add=True)
    visto_por_empleador = models.BooleanField(default=False)
    notas_empleador = models.TextField(blank=True)
```

Listing 5: Modelo Postulacion (fragmento)

4.3. Relaciones Entre Modelos

Modelo 1	Relación	Modelo 2
Usuario	1:1	Empresa
Usuario	1:1	PerfilPostulante
Empresa	1:N	OfertaTrabajo
Categoria	1:N	OfertaTrabajo
OfertaTrabajo	1:N	Postulacion
PerfilPostulante	1:N	Postulacion
Usuario	1:N	Favorito
Usuario	1:N	Notificacion

Cuadro 2: Relaciones entre modelos principales

5. Funcionalidades del Sistema

5.1. Módulo de Autenticación

5.1.1. Registro de Usuarios

El sistema permite el registro de dos tipos de usuarios:

```
def register_view(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')
        nombre = request.POST.get('nombre')
        tipo_usuario = request.POST.get('tipo_usuario')

        # Validaciones
        if Usuario.objects.filter(email=email).exists():
            messages.error(request,
                           'El email ya esta registrado')
            return redirect('register')

        # Crear usuario
        usuario = Usuario.objects.create_user(
            email=email,
            password=password,
            nombre=nombre,
```

```
            tipo_usuario=tipo_usuario
        )

        # Crear perfil segun tipo
        if tipo_usuario == 'postulante':
            PerfilPostulante.objects.create(usuario=usuario)
        else:
            Empresa.objects.create(usuario=usuario)

        messages.success(request, 'Registro exitoso')
        return redirect('login')

    return render(request, 'MyWebApps/register.html')
```

Listing 6: Vista de registro (fragmento)

Proceso de registro:

1. Usuario completa formulario con datos básicos
2. Sistema valida que el email no exista
3. Se crea el usuario con contraseña hasheada
4. Se crea automáticamente el perfil correspondiente (Empresa o PerfilPostulante)
5. Redirección al login

5.1.2. Inicio de Sesión

```
def login_view(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        user = authenticate(request,
                            email=email,
                            password=password)

        if user is not None:
            login(request, user)
            return redirect('dashboard')
        else:
            messages.error(request,
                           'Credenciales invalidas')

    return render(request, 'MyWebApps/login.html')
```

Listing 7: Vista de login (fragmento)

5.2. Módulo de Perfiles

5.2.1. Perfil de Postulante

El postulante puede completar su perfil profesional con:

- Datos personales (nombre, teléfono, ubicación)
- Foto de perfil
- Título profesional
- Resumen profesional
- Nivel de experiencia y años
- Habilidades técnicas
- CV (archivo PDF o DOC)
- Enlaces profesionales (LinkedIn, GitHub, Portafolio)
- Educación
- Experiencia laboral
- Salario esperado
- Disponibilidad

Sistema de Completitud de Perfil

El sistema calcula automáticamente el porcentaje de completitud del perfil basándose en 7 campos obligatorios:

1. Título profesional
2. Resumen profesional
3. Nivel de experiencia
4. Habilidades
5. CV
6. Foto de perfil
7. Al menos un enlace profesional (LinkedIn, GitHub o Portafolio)

La barra de progreso muestra colores dinámicos:

- **Verde**: 70 % o más
- **Naranja**: 40-69 %
- **Rojo**: Menos de 40 %

5.2.2. Perfil de Empresa

El empleador puede configurar su perfil empresarial:

- Nombre de la empresa
- RUC (validado: 11 o 20 dígitos)
- Logo de la empresa
- Descripción
- Sector empresarial
- Ubicación
- Sitio web
- Tamaño de empresa (Startup, PyME, Mediana, Grande)
- Teléfono de contacto

5.3. Módulo de Ofertas de Trabajo

5.3.1. Crear Oferta (Empleador)

```
@login_required
def crear_oferta_view(request):
    if request.user.tipo_usuario != 'empleador':
        messages.error(request,
                       'Solo empleadores pueden crear ofertas')
        return redirect('home')

    empresa = obtener_o_crear_empresa(request.user)

    if request.method == 'POST':
        titulo = request.POST.get('titulo')
        descripcion = request.POST.get('descripcion')
        categoria_id = request.POST.get('categoria')
        # ... m s campos ...

        oferta = OfertaTrabajo.objects.create(
            empresa=empresa,
            categoria_id=categoria_id,
            titulo=titulo,
            descripcion=descripcion,
            aprobada_admin=True,  # Auto-aprobacion
            fecha_publicacion=timezone.now(),
            fecha_expiracion=timezone.now()
                             + timedelta(days=30)
        )

        messages.success(request, 'Oferta publicada')
        return redirect('mis_ofertas')
```

```
29
30      categorias = Categoria.objects.filter(
31          activa=True,
32          eliminado=False
33      )
34      return render(request, 'MyWebApps/crear_oferta.html',
35                    {'categorias': categorias})
```

Listing 8: Crear oferta (fragmento)

Campos de una oferta:

- Título del puesto
- Categoría (Tecnología, Marketing, Ventas, etc.)
- Descripción detallada
- Requisitos
- Responsabilidades
- Beneficios
- Rango salarial (mínimo y máximo)
- Moneda (PEN, USD, EUR)
- Ubicación
- Modalidad (Presencial, Remoto, Híbrido)
- Tipo de contrato (Tiempo completo, Medio tiempo, Por proyecto, Prácticas)
- Nivel de experiencia requerido
- Número de vacantes disponibles

5.3.2. Buscar y Filtrar Ofertas (Postulante)

El sistema permite búsqueda avanzada con múltiples filtros:

```
1  def ofertas_lista_view(request):
2      ofertas = OfertaTrabajo.objects.filter(
3          estado='activa',
4          aprobada_admin=True,
5          fecha_expiracion__gte=timezone.now()
6      )
7
8      # Filtros
9      busqueda = request.GET.get('q')
10     if busqueda:
11         ofertas = ofertas.filter(
12             Q(titulo__icontains=busqueda) |
13             Q(descripcion__icontains=busqueda)
14         )
```

```
15
16      categoria = request.GET.get('categoria')
17      if categoria:
18          ofertas = ofertas.filter(categoria_id=categoria)
19
20      modalidad = request.GET.get('modalidad')
21      if modalidad:
22          ofertas = ofertas.filter(modalidad=modalidad)
23
24      ofertas = ofertas.order_by('-fecha_publicacion')
25
26      return render(request, 'MyWebApps/ofertas_lista.html',
27                    {'ofertas': ofertas})
```

Listing 9: Búsqueda de ofertas (fragmento)

5.4. Módulo de Postulaciones

5.4.1. Postularse a una Oferta

```
1  @login_required
2  def postular_view(request, oferta_id):
3      if request.user.tipo_usuario != 'postulante':
4          messages.error(request,
5                         'Solo postulantes pueden postular')
6          return redirect('home')
7
8      oferta = get_object_or_404(OfertaTrabajo, id=oferta_id)
9      perfil = obtener_o_crear_perfil(request.user)
10
11     # Verificar si ya postulo
12     if Postulacion.objects.filter(
13         oferta=oferta,
14         postulante=perfil
15     ).exists():
16         messages.warning(request, 'Ya postulaste a esta oferta')
17         return redirect('oferta_detalle', oferta_id=oferta_id)
18
19     if request.method == 'POST':
20         carta = request.POST.get('carta_presentacion', '')
21
22         Postulacion.objects.create(
23             oferta=oferta,
24             postulante=perfil,
25             carta_presentacion=carta,
26             estado='pendiente'
27         )
28
29         # Actualizar contador
30         oferta.total_postulaciones = F('total_postulaciones') + 1
31         oferta.save()
32
33         messages.success(request, 'Postulacion enviada')
34         return redirect('mis_postulaciones')
35
```

```
36     return render(request, 'MyWebApps/postular.html',
37                   {'oferta': oferta, 'perfil': perfil})
```

Listing 10: Postular a oferta (fragmento)

5.4.2. Gestionar Postulaciones (Empleador)

El empleador puede ver todas las postulaciones a sus ofertas:

- Lista de postulantes con foto de perfil
- Datos del postulante (nombre, email, teléfono)
- Porcentaje de perfil completado
- Nivel de experiencia
- Disponibilidad
- Badge "Nuevo"si la postulación tiene menos de 48 horas
- Botón para descargar CV
- Opciones para aprobar o rechazar

Estados de postulación:

1. **Pendiente:** Recién enviada
2. **En Revisión:** El empleador la está revisando
3. **Preseleccionado:** Candidato destacado
4. **Entrevista:** Llamado a entrevista
5. **Rechazado:** No seleccionado
6. **Aceptado:** Candidato contratado

5.5. Panel de Administración

5.5.1. Configuración del Admin

Django proporciona un panel administrativo automático que ha sido personalizado:

```
1 from django.contrib import admin
2 from django.utils.html import format_html
3
4 @admin.register(PerfilPostulante)
5 class PerfilPostulanteAdmin(admin.ModelAdmin):
6     list_display = [
7         'usuario',
8         'titulo_profesional',
9         'nivel_experiencia',
```

```
        'progreso_bar',
        'completado_badge'
    ]

    list_filter = [
        'nivel_experiencia',
        'completado',
        'disponibilidad'
    ]

    search_fields = [
        'usuario__email',
        'usuario__nombre',
        'titulo_profesional'
    ]

    def progreso_bar(self, obj):
        porcentaje = obj.porcentaje_completado
        if porcentaje >= 70:
            color = 'green'
        elif porcentaje >= 40:
            color = 'orange'
        else:
            color = 'red'

        return format_html(
            '<div style="width:100px; background:#ddd;">'
            '<div style="width:{}px; background:{}; '
            'text-align:center; color:white;">{}</div>'
            '</div>',
            porcentaje, color, f'{porcentaje}%'
        )

    progreso_bar.short_description = 'Progreso'

    def completado_badge(self, obj):
        if obj.completado:
            return format_html(
                '<span style="background:green; color:white; '
                'padding:3px 10px; border-radius:3px;">'
                'Completo</span>'
            )
        return format_html(
            '<span style="background:orange; color:white; '
            'padding:3px 10px; border-radius:3px;">'
            'Incompleto</span>'
        )

    completado_badge.short_description = 'Estado'

    actions = ['recalcular_porcentaje']

    def recalcular_porcentaje(self, request, queryset):
        for perfil in queryset:
            perfil.calcular_porcentaje_completado()
        self.message_user(request,
```

```
                            'Porcentajes recalculados')

    recalcular_porcentaje.short_description = \
        'Recalcular % completado'
```

Listing 11: Admin personalizado (fragmento)

Características del panel admin:

- Badges de colores para estados
- Barras de progreso visuales
- Filtros por múltiples campos
- Búsqueda avanzada
- Acciones en masa (aprobar, rechazar, etc.)
- Visualización de estadísticas

5.6. API REST

El sistema incluye una API REST completa para integraciones futuras:

5.6.1. Endpoints Principales

Endpoint	Método	Descripción
/api/auth/login/	POST	Login y obtener token
/api/auth/register/	POST	Registrar usuario
/api/ofertas/	GET	Listar ofertas (con filtros)
/api/ofertas/¡id¿/	GET	Detalle de oferta
/api/postulaciones/	GET/POST	Listar/crear postulaciones
/api/categorias/	GET	Listar categorías

Cuadro 3: Endpoints principales de la API

```
from rest_framework import serializers

class OfertaTrabajoSerializer(serializers.ModelSerializer):
    empresa_nombre = serializers.CharField(
        source='empresa.nombre_empresa',
        read_only=True
    )
    categoria_nombre = serializers.CharField(
        source='categoria.nombre',
        read_only=True
    )

    class Meta:
        model = OfertaTrabajo
        fields = [
```

```
            'id', 'titulo', 'empresa_nombre',
            'categoria_nombre', 'descripcion',
            'salario_min', 'salario_max', 'modalidad',
            'ubicacion', 'fecha_publicacion', 'vistas'
        ]
```

Listing 12: Serializer ejemplo

6. Instalación y Configuración

6.1. Requisitos del Sistema

- Python 3.10 o superior
- pip (gestor de paquetes de Python)
- Navegador web moderno
- 500 MB de espacio en disco

6.2. Pasos de Instalación

6.2.1. Paso 1: Clonar el Repositorio

```
git clone https://github.com/JersonCh1/EmpleoyaIW.git
cd EmpleoyaIW
```

6.2.2. Paso 2: Crear Entorno Virtual (Opcional pero Recomendado)

En Windows:

```
python -m venv .venv
.venv\Scripts\activate
```

En Linux/Mac:

```
python3 -m venv .venv
source .venv/bin/activate
```

6.2.3. Paso 3: Instalar Dependencias

```
pip install -r requirements.txt
```

Esto instalará:

- Django 5.2.7
- djangorestframework 3.15.2
- django-cors-headers 4.6.0
- Pillow 11.3.0

6.2.4. Paso 4: Aplicar Migraciones

```
python manage.py migrate
```

Este comando crea todas las tablas en la base de datos SQLite.

6.2.5. Paso 5: Crear Categorías Iniciales

```
python crear_categorias.py
```

Esto crea 10 categorías predefinidas:

1. Tecnología
2. Ventas
3. Marketing
4. Salud
5. Educación
6. Finanzas
7. Legal
8. Construcción
9. Gastronomía
10. Otros

6.2.6. Paso 6: Crear Usuario Administrador

```
python manage.py createsuperuser
```

Ingresa:

- Email: `admin@empleoya.com`
- Nombre: `Admin`
- Contraseña: (la que prefieras)

6.2.7. Paso 7: Ejecutar el Servidor

```
python manage.py runserver
```

El sistema estará disponible en: http://127.0.0.1:8000/

6.3. Configuración Adicional

6.3.1. Variables de Entorno (Producción)

Para despliegue en producción, configurar:

```
# En settings.py
DEBUG = False
ALLOWED_HOSTS = ['tu-dominio.com']

# Base de datos PostgreSQL
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'empleoya_db',
        'USER': 'postgres',
        'PASSWORD': 'tu_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

7. Pruebas y Validación

7.1. Casos de Prueba

7.1.1. CP001: Registro de Usuario Postulante

Objetivo: Verificar que un postulante puede registrarse correctamente.
Precondiciones:

- Sistema en funcionamiento
- Email no registrado previamente

Pasos:

1. Navegar a `/register/`
2. Ingresar datos válidos
3. Seleccionar tipo: "Postulante"
4. Hacer clic en Registrarse"

Resultado esperado:

- Usuario creado exitosamente
- PerfilPostulante creado automáticamente
- Redirección a página de login
- Mensaje de confirmación

7.1.2. CP002: Completar Perfil al 100 %

Objetivo: Verificar el cálculo del porcentaje de completitud.
Pasos:

1. Login como postulante
2. Ir a `/perfil/postulante/`
3. Completar todos los campos obligatorios
4. Guardar cambios

Resultado esperado:

- Barra de progreso muestra 100 %
- Color verde en la barra
- Mensaje de perfil completo

7.1.3. CP003: Crear Oferta de Trabajo

Objetivo: Verificar que un empleador puede publicar ofertas.
Precondiciones:

- Usuario de tipo .Empleador"
- Categorías creadas en el sistema

Pasos:

1. Login como empleador
2. Ir a `/crear-oferta/`
3. Completar formulario
4. Hacer clic en "Publicar"

Resultado esperado:

- Oferta creada con `aprobada_admin=True`
- Visible inmediatamente en búsqueda
- Aparece en "Mis Ofertas"

7.1.4. CP004: Postularse a Oferta

Objetivo: Verificar el proceso de postulación.
Pasos:

1. Login como postulante
2. Buscar oferta
3. Hacer clic en "Postular"
4. Escribir carta de presentación (opcional)
5. Confirmar postulación

Resultado esperado:

- Postulación creada con estado "Pendiente"
- Contador de postulaciones incrementado
- Aparece en "Mis Postulaciones"
- No puede volver a postular a la misma oferta

7.2. Validaciones de Seguridad

7.2.1. Protección CSRF

Django incluye protección contra Cross-Site Request Forgery:

```
<form method="POST">
    {% csrf_token %}
    <!-- campos del formulario -->
    <button type="submit">Enviar</button>
</form>
```

Listing 13: Token CSRF en formularios

7.2.2. Autenticación Requerida

Vistas protegidas con decorador:

```
from django.contrib.auth.decorators import login_required

@login_required
def perfil_postulante_view(request):
    # Solo usuarios autenticados pueden acceder
    pass
```

7.2.3. Validación de Permisos

```python
def crear_oferta_view(request):
    if request.user.tipo_usuario != 'empleador':
        messages.error(request,
                       'No tienes permisos')
        return redirect('home')
    # ... resto del codigo
```

8. Características Avanzadas

8.1. Sistema de Búsqueda Inteligente

El sistema implementa búsqueda con múltiples criterios:

```python
from django.db.models import Q

ofertas = OfertaTrabajo.objects.filter(
    Q(titulo__icontains=busqueda) |
    Q(descripcion__icontains=busqueda) |
    Q(requisitos__icontains=busqueda),
    estado='activa',
    aprobada_admin=True
)
```

Listing 14: Búsqueda avanzada

8.2. Diseño Responsive

El sistema utiliza CSS Grid y Flexbox para adaptarse a diferentes dispositivos:

```css
/* Mobile First */
.grid {
    display: grid;
    grid-template-columns: 1fr;
    gap: 1rem;
}

/* Tablet */
@media (min-width: 768px) {
    .grid-2 {
        grid-template-columns: repeat(2, 1fr);
    }
}

/* Desktop */
@media (min-width: 1024px) {
    .grid-3 {
        grid-template-columns: repeat(3, 1fr);
    }
}
```

Listing 15: CSS responsive (fragmento)

8.3. Avatares Dinámicos

Si el usuario no tiene foto, se muestra su inicial con gradiente:

```
<div class="user-info">
    {% if user.foto_perfil %}
        <img src="{{ user.foto_perfil.url }}"
             class="user-avatar">
    {% else %}
        <div class="user-avatar-placeholder">
            {{ user.nombre|first|upper }}
        </div>
    {% endif %}
    <span class="user-name">{{ user.nombre }}</span>
</div>
```

Listing 16: Avatar con placeholder

```
.user-avatar-placeholder {
    width: 36px;
    height: 36px;
    border-radius: 50%;
    background: linear-gradient(135deg, #3b82f6, #10b981);
    color: white;
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: bold;
}
```

8.4. Badge "Nuevo."en Postulaciones

Las postulaciones recientes (menos de 48 horas) se destacan:

```
{% if postulacion.fecha_postulacion|timesince < "48 horas" %}
    <span class="badge badge-success">Nuevo</span>
{% endif %}
```

8.5. Helpers para Prevenir Errores 404

```
def obtener_o_crear_perfil(usuario):
    """Obtiene o crea perfil de postulante"""
    try:
        return PerfilPostulante.objects.get(
            usuario=usuario
        )
    except PerfilPostulante.DoesNotExist:
        return PerfilPostulante.objects.create(
            usuario=usuario
        )

def obtener_o_crear_empresa(usuario):
    """Obtiene o crea empresa para empleador"""
    try:
```

```
        return Empresa.objects.get(usuario=usuario)
    except Empresa.DoesNotExist:
        return Empresa.objects.create(
            usuario=usuario,
            nombre_empresa=f"Empresa de {usuario.nombre}"
        )
```

Listing 17: Helper functions

9. Mejores Prácticas Implementadas

9.1. Código Limpio

- Nombres descriptivos de variables y funciones
- Funciones pequeñas con responsabilidad única
- Comentarios donde es necesario
- Código DRY (Don't Repeat Yourself)

9.2. Seguridad

- Contraseñas hasheadas con PBKDF2
- Protección CSRF en todos los formularios
- Validación de permisos en vistas
- Sanitización de inputs
- Prevención de SQL Injection con ORM

9.3. Performance

- Uso de `select_related()` y `prefetch_related()`
- Índices en campos frecuentemente buscados
- Paginación en listados largos
- Carga lazy de imágenes

9.4. Mantenibilidad

- Estructura modular
- Separación de concerns (MVT)
- Uso de helpers y utilidades
- Documentación inline

10. Trabajo Futuro

10.1. Mejoras Propuestas

10.1.1. Corto Plazo

- Sistema de notificaciones en tiempo real (WebSockets)
- Chat integrado entre empleador y postulante
- Exportación de reportes a PDF/Excel
- Sistema de calificaciones y reseñas
- Alertas de empleo personalizadas

10.1.2. Mediano Plazo

- Integración con LinkedIn para importar perfiles
- Videollamadas integradas para entrevistas
- Sistema de recomendaciones con Machine Learning
- Tests automatizados con pytest
- Dashboard de analíticas avanzadas

10.1.3. Largo Plazo

- Aplicación móvil nativa (iOS/Android)
- Sistema de pagos para planes premium
- Marketplace de servicios freelance
- Integración con ATS (Applicant Tracking Systems)
- Expansión internacional multi-idioma

10.2. Escalabilidad

Para soportar mayor tráfico:

- Migrar a PostgreSQL
- Implementar Redis para caché
- Usar Celery para tareas en segundo plano
- CDN para archivos estáticos
- Load balancing con Nginx
- Containerización con Docker

11. Conclusiones

11.1. Logros del Proyecto

1. Se desarrolló exitosamente una plataforma web funcional de bolsa de trabajo con Django 5.2.7
2. Se implementaron dos roles de usuario (Postulante y Empleador) con funcionalidades específicas y dashboards personalizados
3. Se creó un sistema de perfiles profesionales con cálculo automático de completitud, alcanzando el 100 % al llenar 7 campos obligatorios
4. Se implementó un módulo CRUD completo para ofertas de trabajo con búsqueda y filtrado avanzado
5. Se desarrolló un sistema de postulaciones con 6 estados diferentes y gestión visual para empleadores
6. Se personalizó el panel administrativo de Django con badges, barras de progreso y acciones en masa
7. Se implementó una API REST documentada para futuras integraciones
8. Se aplicaron mejores prácticas de seguridad (CSRF, autenticación, validación de permisos)
9. Se logró un diseño responsive que se adapta a móviles, tablets y desktop
10. Se implementaron características UX avanzadas (avatares dinámicos, badges "Nuevo", hover effects)

11.2. Aprendizajes

Durante el desarrollo del proyecto se adquirieron conocimientos en:

- Arquitectura MVT de Django
- ORM de Django para abstracción de base de datos
- Sistema de autenticación y permisos
- Django REST Framework para APIs
- Manejo de archivos subidos (CVs, fotos, logos)
- Personalización del Django Admin
- Diseño responsive con CSS Grid y Flexbox
- Validaciones del lado del servidor
- Gestión de estados en aplicaciones web

11.3. Impacto del Proyecto

EMPLEOYA demuestra la viabilidad de crear plataformas web profesionales utilizando tecnologías open-source como Django. El sistema implementado podría ser utilizado por:

- Universidades para conectar estudiantes con empresas
- Empresas de reclutamiento para gestionar vacantes
- Municipalidades para bolsas de trabajo locales
- Organizaciones gremiales para oportunidades sectoriales

11.4. Reflexión Final

El proyecto EMPLEOYA cumplió exitosamente con los objetivos planteados, entregando una plataforma funcional, segura y escalable. La experiencia adquirida en el desarrollo full-stack con Django sienta las bases para proyectos web más complejos en el futuro.

La elección de Django como framework principal resultó acertada debido a su filosofía "batteries included", que permitió desarrollar rápidamente funcionalidades complejas sin sacrificar seguridad ni mantenibilidad.

12. Referencias

1. Django Documentation. (2025). *Django Web Framework*. https://docs.djangoproject.com/
2. Django REST Framework. (2025). *Web APIs for Django*. https://www.django-rest-framework.org/
3. Mozilla Developer Network. (2025). *HTML, CSS & JavaScript*. https://developer.mozilla.org/
4. Python Software Foundation. (2025). *Python Documentation*. https://docs.python.org/3/
5. Pillow Documentation. (2025). *Python Imaging Library*. https://pillow.readthedocs.io/
6. GitHub Repository. (2025). *EMPLEOYA Project*. https://github.com/JersonCh1/EmpleoyaIW

A. Apéndice A: Comandos Útiles

A.1. Gestión de Migraciones

```
# Crear migraciones
python manage.py makemigrations

# Aplicar migraciones
python manage.py migrate

# Ver estado de migraciones
python manage.py showmigrations

# Revertir migracion especifica
python manage.py migrate MyWebApps 0001
```

A.2. Gestión de Base de Datos

```
# Abrir shell de Django
python manage.py shell

# Acceder a base de datos SQL
python manage.py dbshell

# Cargar datos desde fixture
python manage.py loaddata datos.json

# Exportar datos a fixture
python manage.py dumpdata MyWebApps > datos.json
```

A.3. Gestión de Usuarios

```
# Crear superusuario
python manage.py createsuperuser

# Cambiar password de usuario
python manage.py changepassword usuario@email.com
```

B. Apéndice B: Estructura de requirements.txt

```
# Dependencias esenciales para EMPLEOYA
# Instalar con: pip install -r requirements.txt

# Framework web
Django==5.2.7

# API REST
djangorestframework==3.15.2

# CORS para API
django-cors-headers==4.6.0

# Manejo de imagenes (fotos de perfil, logos, CVs)
```

```
Pillow==11.3.0
```

Listing 18: requirements.txt completo

C. Apéndice C: Configuración de settings.py

```python
# Settings principales
DEBUG = True
ALLOWED_HOSTS = []

# Base de datos
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Archivos media
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Archivos estaticos
STATIC_URL = '/static/'
STATIC_ROOT = BASE_DIR / 'staticfiles'

# Autenticacion
AUTH_USER_MODEL = 'MyWebApps.Usuario'
LOGIN_URL = 'login'
LOGIN_REDIRECT_URL = 'dashboard'
LOGOUT_REDIRECT_URL = 'home'

# Seguridad
CSRF_COOKIE_HTTPONLY = True
SESSION_COOKIE_HTTPONLY = True
SECURE_BROWSER_XSS_FILTER = True
```

Listing 19: Configuraciones importantes

D. Apéndice D: Glosario de Términos

API REST Application Programming Interface basada en arquitectura REST para comunicación entre sistemas

CRUD Create, Read, Update, Delete - Operaciones básicas de persistencia

Django Framework web de Python de alto nivel que fomenta el desarrollo rápido

MVT Model-View-Template - Patrón arquitectónico de Django

ORM Object-Relational Mapping - Mapeo objeto-relacional para abstracción de BD

Slug Versión URL-friendly de un texto (ej: "desarrollador-full-stack")

Template Plantilla HTML con lógica de Django para renderizar contenido dinámico

Middleware Componente que procesa peticiones y respuestas globalmente

Migration Archivo que describe cambios en la estructura de la base de datos

Serializer Componente que convierte objetos Python a JSON (y viceversa)

E. Apéndice E: Contacto y Soporte

E.1. Autores

Piero De La Cruz

- Universidad La Salle
- Ingeniería de Software

Jerson Chura

- Universidad La Salle
- Ingeniería de Software

E.2. Repositorio del Proyecto

- GitHub: https://github.com/JersonCh1/EmpleoyaIW
- Issues: https://github.com/JersonCh1/EmpleoyaIW/issues

E.3. Documentación Adicional

Dentro del repositorio se encuentran:

- `README.md` - Documentación principal
- `GUIA_PROYECTO.md` - Guía paso a paso
- Comentarios inline en el código